LA-UR -89-2706

LA-UR--89-2706

DE89 016769

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405 ENG 36

TITLE An Integrated Distributed Processing Interface for Supercomputers and Workstations

AUTHOR(S) John R. Campbell and Lauren P. McGavran

SUBMITTED TO ASE 89
Applications of Supercomputers in Engineering
Southampton University, United Kingdom
September 5 - 7, 1989

An Integrated Distributed Processing Interface for Supercomputers and Workstations

John Campbell and Lauren McGavran

Los Alamos National Laboratory

Computer Research and Applications

Los Alamos, New Mexico 87544

United States of America

Abstract

Access to documentation, communication between multiple processes running on heterogeneous computers, and animation of simulations of engineering problems are typically weak in most supercomputer environments. This presentation will describe how we are improving this situation in the Computer Research and Applications group at Los Alamos National Laboratory.

We have developed a tool using UNIX filters and a SunView interface that allows users simple access to documentation via mouse driven menus. We have also developed a distributed application that integrates a two point boundary value problem on one of our Cray Supercomputers. It is controlled and displayed graphically by a window interface running on a workstation. Finally, we will describe software to animate results on the workstation screen. Our motivation for this research has been to improve the usual typewriter/static interface using language independent controls to show capabilities of the workstation supercomputer combination.

Introduction

At the Los Alamos National Laboratory we have a vast array of computing hardware[1]. While this hardware presents an incredible potential to the many scientists, mathematicians, and engineers who use our facilities, it also presents special problems in terms of interconnections between the various computers and usage of our facilities. Because of the number of different computer languages and operating systems it is difficult for our users to control the flow of information in their applications. Compounding these difficulties are the different languages used in our various disciplines.

We are continually confronted by hardware and software that is either too simplistic for a given application, or that is language bound (i.e. its effective use is inhibited by voluminous verbiage). An example of this language barrier is the documentation we have all faced when presented with new capabilities. Volumes of instruction manuals do not compensate for simple pictorial examples of available tools and their interrelationships[2].

Another problem we all face is our access to tools like the SLATEC[3] math library. This library is composed of several thousand routines by many authors and each author uses his own terminology for common objects like dependent variables, work arrays, and so forth. The reference manual explaining each author's terms for these objects is usually contained in a separate compendium. What we really want to have is an on-line hierarchical documentation tool that would allow us to search for appropriate programs for the task at hand and give us examples of each routine's usage. We would like this search to occur with a minimum of typing, and without having to learn the documentation tool syntax in addition to the variable notation of the math tool.

After deciphering the math tool semantics to get us to our goal, we want to connect our computers synergistically. In the case of trying to solve partial differential equations, for example, the workstation should be used for the user interface and not to integrate the pde's. Similarly, the supercomputer should be used to integrate the pde's and not be concerned with the details of the user interface. By communicating between processes running on each, we are able to achieve the compute power of the supercomputer controlled by a friendly interface possible with most modern workstations.

Having overcome the semantic difficulties and distributed the processes to heterogeneous computers, we want to use the full capability of our workstation to control the problem's parameters. We want to use analog type controls, color, three dimensional representations, and animation to gain insight into our problems. Only then can we apply our full intuition to the interpretation of the numerical models.

This paper describes how we have demonstrated these concepts in practical applications.

Tool Access

DOC is a program written in the C programming language that implements a mouse to extract documentation for a particular subroutine from a library of mathematical software. It is foolproof in the sense that its user has only to select from a series of menus. These menus are activated by pushing buttons with the name of the particular library, such as SLATEC. This "mouse click" activates a menu of the names of categories contained in that library. When the mouse is moved to the right of a particular category, its corresponding subcategory menu pops up. Activating the subcategory's pullright menu causes its corresponding subroutine menu to pop up. Selection of a particular routine by clicking the mouse over the name of that routine causes its description to appear in a section of the window that can be easily scrolled up and down. Thus the user of DOC has a simple, responsive method for understanding the appropriate usage for routines in that library.

The way that DOC works is that the menus for the category, subcategory and routine menus are built using the "awk" Unix filter[4]. The category and subcategory menus are built when DOC is started, and the routine menu is built "on the fly". That is, the contents of the routine menu is generated according to what has been selected from the first two menus. The program works by starting up processes at the appropriate times to yield routine names for what is selected from the previous menu items.

The trick for extracting the appropriate routine description quickly is random access to the file that contains the descriptions of all the routines in the library. The "tail" Unix filter is used to go directly to the starting byte location for the routine chosen and the "extract" shell script, which contains tail, extracts the documentation for the routine selected by the user.

The "extract" shell script requires that the documentation file containing documentation for all the routines be preprocessed into a linked list of subroutine names, followed by a starting byte location and the number of lines of documentation per routine. Although this sounds complicated, it is easily accomplished with a 3 line shell script that also relies on the "awk" Unix filter.

DOC is useful in itself and is an example of how the power and flexibility of Unix filters can be integrated into programs and windowing interfaces. This could be very useful in numerical analysis where we often need to sort or filter our data and pass the results from one process to another in as simple a manner as possible.

DOC does simple interprocess communication but the Berkeley version of Unix[5] has a concise and efficient means of communicating between multiple processes running on the same or different CPUs. The interprocess communication or IPC tools allow us to share memory and exchange messages between processes, and to synchronize processes through semaphores. Thus, in the particular example of DOC, the latency time of several seconds needed to start up the process that generates the subroutine menu could be improved by having a separate program running. This separate program would generate the category and subcategory menus and then go into a wait state until a signal was passed asking it to wake up. It would then read the message stack to determine what set of menu items the user had selected. Finally it would generate the appropriate subroutine names and leave these on the menu stack of messages tagged to keep the communications straight and go back to sleep. When the appropriate semaphore was set indicating that it was done, the window managing program would read the message stack for appropriately tagged items and activate the correct menu for the user to choose from.

Use of IPC's in numerical analysis has yet to be exploited even though there are many areas where multiple processes could improve simulations. Even less utilized in application research is distributed processing, which we describe next. Distributed processing means using multiple processes similar to IPC's but splits them among heterogeneous computers connected by an Ethernet or bus connection to take advantage of special hardware such as vector registers and 3 dimensional geometry engines.

Distributed Processing

Supercomputers and workstations can be coupled synergistically so that we have the compute power of the supercomputer controlled by a smart interface. Analog controls such as buttons and sliders give users prompt feedback freeing their imagination and intuition for their basic problems as opposed to the linguistics of how to attain the information they desire.

There are three ways of distributing processes between computers. The most recognizable to those who program in Fortran is the Remote Procedure Call or RPC. Using RPC's, a server program is started on the remote machine. Then the subroutines in that program can be called from a client program running on the local machine as if the routines in the remote file were linked into the client.

If the two computers have different internal byte representations for floats, integers, and other variable types, they must be interpreted so that each machine refers to the same numerical value for the same variable. The most common technique for this data correction is the external data representation or XDR. Using XDR's, the variables are interpreted in IEEE standard format before they are sent over the network to the other computer. At the receiving end the data is then deciphered in the native byte representation of that computer.

RPCODE is an example of the use of a workstation coupled to a supercomputer. It uses a supercomputer to integrate the ordinary differential equation (ODE) shown below between two boundary values: yleft and yright:

eps y'' + x y' - y = 0

Once the server and client are running and connected, the solution to this equation is returned whenever the user adjusts the yleft, yright, or eps sliders. These sliders are in a control panel subwindow within a window that also contains a canvas subwindow. When the endpoints of the vectors that represent the solution are returned the solution is drawn in the canvas subwindow

The RPC's described above allow us to do remote execution between computers using familiar concepts (i.e. the procedure call). RPC's are built on top of another Unix tool known as sockets. Sockets are the basis for name servers, file servers, ftp, etc. They are endpoints of communication between processes. They allow processes to communicate over ports; the processes can be on different machines or on the same machine. Thus communication paths may be controlled by software. This method of communication also allows a user to "select" on a particular process/port number. What select means is that the read or write from one process can determine if the corresponding write/read on the other process is ready to send or receive data

The select sy tem call is passed a bit mask that is composed of "on" bits for the open file descriptors, and it returns a bit mask that represents the socket descriptors that are waiting to complete their i/o. Thus processes on different computers may be multiplexed. It is an efficient way of controlling communication by software. There are several types of sockets but the most important for applications are the user datagram protocol (UDP) and the transmission control protocol (TCP).

UDP sockets use minimal overhead and are fast but unreliable, whereas TCP sockets guarantee that information sent will arrive and be in the right order independent of the physical route taken. Sockets can also send data out of band (OOB). OOB data may be signals, interrupts, synchronization parameters, and so forth. OOB data shows up as fast as possible, independent of the state of other data being sent over a socket descriptor.

To use sockets one must learn about internet addresses and how the C programming language handles i/o (which is usually more complicated than Fortran i/o). A simpler mechanism for getting from a process on one machine to a process on another machine is the REXEC. REXEC is a system call that starts up a process on a remote machine from a C program running on a local machine and connects the standard input and output of the remote process to the socket descriptor returned by the REXEC call.

The simplicity and small amount of code needed for doing RPC's, IPC's, and REXEC's means code can be modular and flexible and therefore the minimal software for a given functionality. This is one of the most important features of Unix. The motivation for this brevity is that Unix was originally developed on a teletype. Still minimal verbiage/code to do what one wants to do, and features like name hiding (available in all window systems) allow us to make our programs more flexible and structured. Numerical analysis can also be advanced by concepts analogous to Unix handles files and processes. Processes have life. That is they come into existence, have their own separate instruction set separate from other processes, and run independently of other processes. Similarly files have position and space. Although a file may reside on many different disks, it has a simple logical name/inode address so it may be referred to as a contiguous entity. Unix accomplishes this homogeneity by using the powerful concept of pointers[5]

Animation

Fast computers should also be able to produce fast images. Animation can help give us insight into the time dependent aspects of our problems. However, animation is a often very difficult. This difficulty is because doing things quickly in Unix often means learning how the kernel controls the flow of information, which drops us to the device driver level of software

Many of us have generated film images of mathematics/engineering to show on a movie projector. Projectors generally run at 24 frames/second so the interpolation to get real time is a simple algebraic problem. This method is not interactive since we must wait for the film to be developed to see what we've generated.

We can do a more controllable animation on raster based workstations because of their refresh capability. Motion can be produced by writing successive images into memory and copying these images from memory to the screen using the polling function. We have produced software to make a sine wave move across the screen in this way. Frequency and amplitude are controlled by sliders that change the appropriate images in memory when they are adjusted by the user. The lines that represent the sine wave are drawn into their corresponding memory locations and another procedure that copies these images to the screen is called as often as possible using the polling function. Thus the sine wave appears to move across the window

This form of animation is limited to rectangular regions of the screen because of the memory to screen mapping software. True random pixel access can be achieved by opening the workstation's frame buffer device driver, allocating virtual memory using VALLOC, and mapping a variable to that memory representing the screen memory using MMAP. Then as the variable is changed, corresponding pixel locations are changed. There are no reads or writes and so screen changes happen at memory rates

Conclusion

In conclusion, we have shown how IPC's, distributed processing, and Unix filters can make numerical analysis more productive. In the future we hope to use an object oriented language like C++ [6] to make libraries that simplify communication and synchronization of processes so everyone does not have to learn the lowest level communications in order to do his mathematics on heterogeneous machinery connected by either Ethernet or high speed buses Distributed operating systems like DUNIX [7] will play an increasingly important role in making distributed processing more language transparent and visual programming control environments lik VOUS [8] will improve the long learning curve now necessary in our varied programming environment

We eventually hope to solve problems like fluid mechanics on distributed processors fast enough to be able to show evolutionary trends. Using the techniques described in this paper we can potentially solve coupled partial differential equations by distributing the individual equations solvers to separate CPU's. Shared boundary can be communicated at a sustained nominal rate of 1 megabit/sec. and synchronization signals can be passed around so that one computer controls all the integration and communication activity. In order to make the data recognizable we can send the data to yet another processor that is taking the physical variables, such as pressure and the 3 components of velocity, and doing further mathematical processing to produce information in a form that may be assimilated into a hardware graphics engine capable of rendering the visual images in real time.

This hardware has the capability to render 3 dimensional surfaces and rotate them quickly using a knob box. Therefore, in the near future we hope to be able to recognize where our numerics need more resolution by customized representations of derived quantities such as the divergence and vorticity of a flow.

By appropriate communication between computers and people we can achieve many interesting applications with the hardware tools becoming available at an ever increasing rate. Without this communication, however, we will continue to run nothing more innovative than benchmark routines on computers with speeds approaching 100 megaflops or pictures of teapots on high powered graphics engines capable of rendering variably transparent 3-dimensional surfaces with light sources in real time.

REFERENCES

1 Sander, Janet and Garnet, Ann, Computing and Communications Division News, July 1989.

2 Mckim, Robert H., Thinking Visually, A Strategy Manual for Problem Solving, Lifetime Learning Publications, Beaumont, California, 1980.

3 Buzzbee, Bill L. "The SLATEC Common Math Library" in Sources and Development of Mathematical Software, W. Cowl, Red., Prentice-Hall Series in Computational Mathematics.

4 Kernighan, Brian W. And Pike, Rob, The UNIX Programming Environment, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1984.

5 Bach, Maurice J., The Design of the UNIX Operating System, Prentice-Hall Software Series, New Jersey, 1986.

6 Stroustrap, Bjarne, The C++ Programming Language, Addison-Wesley Publishing Company, Menlo Park, California, 1987.

7 Litman, Ami, The DUNIX Distributed Operating System, Bell Communications Research

8 Oliger, Joseph, Pichumani, Ramani, and Ponceleon, Dulce, A Visual Object-Oriented Unification System, Stanford University Manuscript CLaSSiC-89-23, 1989.